

# FAA National Software Conference, June 2001

## Software Reliability

### Software Reliability

Is there anything to it besides a set of hardware-derived models?

Software Reliability  
© Jeff Knickerbocker, 2001

1

SW DER Conference 2001

### Software Reliability

*2.2.3 Software Level Determination, 12.3.4 Software Reliability Models (FAQ #26)*

#### Software Level Determination

- Development of software to a software level does not imply the assignment of a failure rate for that software. Thus, software levels or **software reliability rates based on software levels cannot be used by the system safety assessment process as can hardware failure rates.**
- Strategies which depart from the guidelines of this paragraph (2.2.3) need to be justified by the system safety assessment process.

#### Software Reliability Models

- During the preparation of this document, methods for estimating the post-verification probabilities of software errors were examined. The goal was to develop numerical requirements for such probabilities for software in computer-based airborne systems or equipment. The conclusion reached, however, was that **currently available methods do not provide results in which confidence can be placed to the level required for this purpose.** Hence, this document does not provide guidance for software error rates.
- **If the applicant proposes to use software reliability models for certification credit, rationale for the model should be included in the Plan for Software Aspects of Certification, and agreed with by the certification authority.**



Software Reliability  
© Jeff Knickerbocker, 2001

2

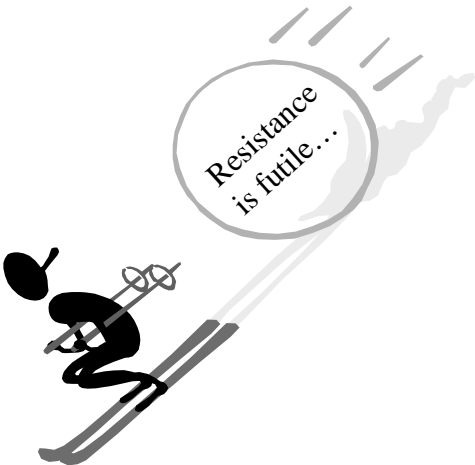
SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Software Reliability

Is it worth considering software reliability?



- The current regulatory guidance does not **prohibit** the use of reliability models
- DO-178B is not **required** by the FARs...
- Just what is the concern with using reliability models? Isn't reliability a "good thing"?
- Can we learn anything from a quick look at reliability concepts for either hardware or software?

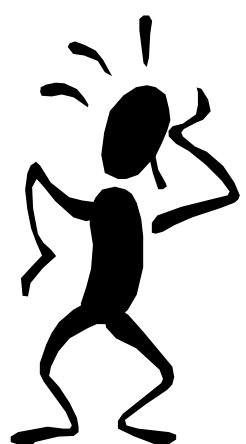
Software Reliability  
© Jeff Knickerbocker, 2001

3

SW DER Conference 2001

### Reliability

Reliability is ...



**reliability** – the **probability** that a given item will perform its **required function** under given **conditions** for a stated **time interval**

- Hardware components usually have published, known failure characteristics under operation – in short **a history** on which failure predictions may be derived
- **New** software components have no known history in an integrated system under specific conditions - software components are **typically** custom components
- Hardware will eventually wear out - **executable** software is a series of binary patterns that will never wear out per se
- Hardware failure categories are typically treated as **random faults** (statistical methods may be applied) - software faults are **design faults** that may not be random (can statistical methods be applied?)

Software Reliability  
© Jeff Knickerbocker, 2001

4

SW DER Conference 2001

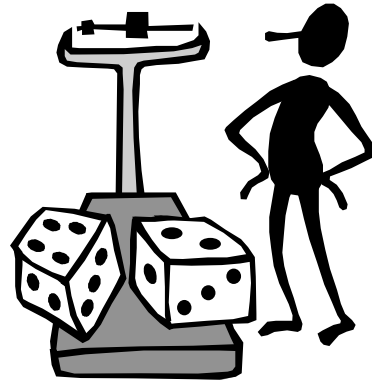
# FAA National Software Conference, June 2001

## Software Reliability

### Software Faults - Random?

“Both the human error process that introduces defects into code and the run selection process that determines which code is being executed at any time are dependent on an enormous number of time-varying variables. The use of a random process model is appropriate for such a situation.”

-John Musa



Software Reliability  
© Jeff Knickerbocker, 2001

5

SW DER Conference 2001

### Reliability Models - Hardware

Several common models...

- Exponential

Density function	$f(t) = \lambda e^{-\lambda t}$
Reliability function	$R(t) = e^{-\lambda t}$
Failure rate	$\lambda(t) = \lambda$
MTBF	$1 / \lambda$

- Weibull

Density function	$f(t) = (\beta/\eta)[(t - \gamma)/\eta]^{(\beta - 1)}e^{-[(t - \gamma)/\eta]^\beta}$
Reliability function	$R(t) = e^{-[(t - \gamma)/\eta]^\beta}$
Failure rate	$\lambda(t) = (\beta/\eta)[(t - \gamma)/\eta]^{(\beta - 1)}$
MTBF	$t_{\text{bar}} = \gamma + \eta\Gamma(\beta^{-1} + 1)$

Software Reliability  
© Jeff Knickerbocker, 2001

6

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Reliability Models - Software

Model de jour...

- Models, models everywhere...
  - New software reliability models are presented in *IEEE Transactions on Software Engineering* almost monthly
  - Numerous publications are available on software reliability (a search of Amazon for titles on software reliability yielded 145 titles)
  - Some are models very complex (Kalman filter-like adaptive processes) while others are quite simple
  - Most models tend to be some type of exponential function

Software Reliability  
© Jeff Knickerbocker, 2001

7

SW DER Conference 2001

### Three Basic Software Reliability Models

- Simple Ratio Model
- Musa Exponential Model
- Musa-Okumoto Logarithmic Model (ref only)

These examples were chosen due to their relative simplicity and the widespread availability of supporting information for those interested in software reliability models.

There are many, many, possibly better choices - no specific model is being advocated!

Software Reliability  
© Jeff Knickerbocker, 2001

8

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Simple Ratio Model

Using failure ratios to determine MTBF and Reliability

- This model is defined as follows:

MTBF =  $t/r$  where

$t$  = cumulative running time and

$r$  = total number of failures noted

$R(T) = e^{-\lambda T}$  where

$\lambda = 1/\text{MTBF}$

$T$  is the mission duration

Software Reliability  
© Jeff Knickerbocker, 2001

9

SW DER Conference 2001

### Simple Ratio Model

Example

Assume a software manufacturer has developed a software intensive system, performed formal verification and validation activities, and placed the system in the field. After 50,000 hrs of operation (run time versus calendar time) 25 independent errors have been reported by the users of the system. Then the MTBF of the system would be determined as follows:

$\text{MTBF} = t/r = 50,000 \text{ hr}/25 \text{ failures} = 2000 \text{ hr/failure}$

While the reliability of the same system for 100 hours would be calculated as shown:

$R(100) = e^{-(100/2000)} \sim 0.95122$  or 95%

or, reliability could be determined from the graph on the following page:

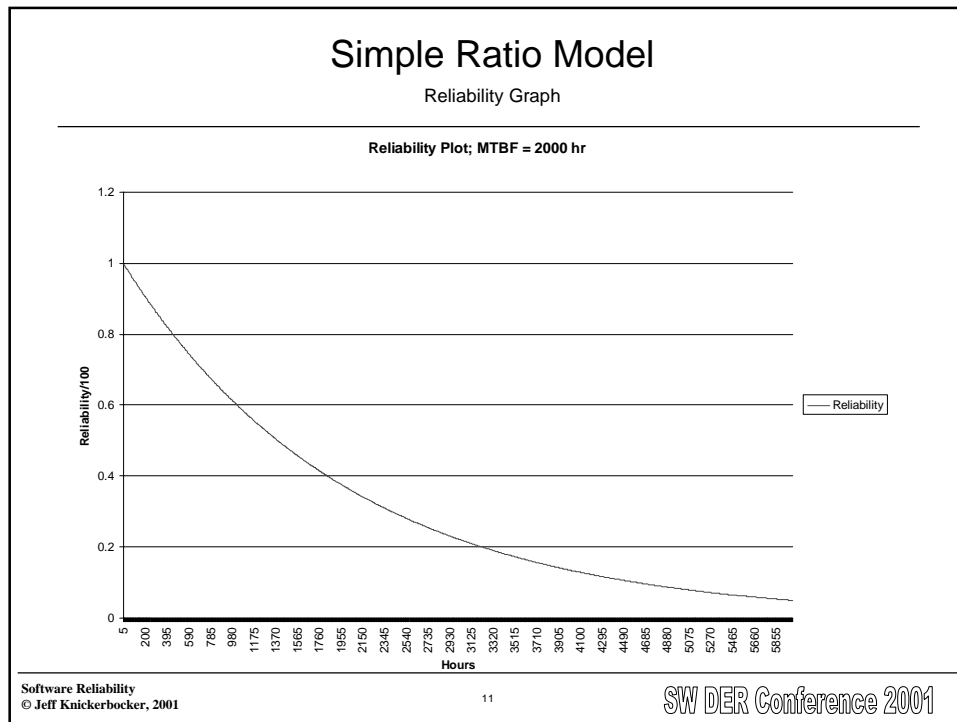
Software Reliability  
© Jeff Knickerbocker, 2001

10

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability



**Simple Ratio Model**  
Strengths

---

- Simple to calculate
- Simple to understand
- Similar to the hardware exponential model
- Use for reliability growth development if adequate failure reporting systems are in place
- Useful for comparing software intensive products for identical missions assuming there are many, many hours of service history (e.g., two different suppliers of a TSO'd device)
- Fundamentally based on observations versus statistical predictions

Software Reliability  
© Jeff Knickerbocker, 2001

12

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Simple Ratio Model

Concerns

- When does the evaluation start?
  - After preliminary debug?
  - After a full verification and validation?
  - Applying the model too early may provide a false indicator of poor reliability
- What is defined as an error versus “future enhancement”?
  - Unless defined *a priori*, suppliers may define errors in a way that puts them in the best light
- How will time statistics be collected?
  - User’s estimates may vary without an embedded clock
  - Does the clock restart if the software modified?
  - A “fix” may actually create additional problems...
- How is it ensured that the user’s environment(s) are similar to the supplier’s test environment?
- How are errors categorized? Unless counted carefully, errors may be counted multiple times

Software Reliability  
© Jeff Knickerbocker, 2001

13

SW DER Conference 2001

### Musa Models - Terminology

- failure intensity
  - Failures per natural or time unit; represented by  $\lambda$
- fault
  - A defect in the system that causes a failure when executed.  
A software fault is a defect in the code
- operational profile
  - A set of operations and their probabilities of occurrence  
(often in the form of a test requirements suite)

Software Reliability  
© Jeff Knickerbocker, 2001

14

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Potential Applications

- Failure intensity versus the total number of failures experienced
- Experienced failures versus time
- Failure intensity versus time
- Expected failures to reach a specific fault intensity (reliability growth)
- Expected time to reach a specific fault intensity (reliability growth)
- Field failure intensity

Software Reliability  
© Jeff Knickerbocker, 2001

15

SW DER Conference 2001

### Musa Exponential Model

Key Parameters and Fundamental Equation

- Basic Equation

$$\lambda(T) = \lambda_0 e^{(-\lambda_0 T / v_0)}$$

←Note the similarity to the HW version

- $\lambda$  failure intensity
- $\lambda_0$  initial failure intensity (empirically determined)
- $v_0$  total failures in infinite time (empirically determined)
- $\mu$  average or expected number of failures at a given point in time
- $T$  time duration for which an estimation or prediction is being made in **execution time**

Software Reliability  
© Jeff Knickerbocker, 2001

16

SW DER Conference 2001



# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Failure Intensity vs. Total Number of Failures

- This form of this equation is defined as follows:

$$\lambda(\mu) = \lambda_0(1 - \mu/v_0)$$

- Note the slope of the failure intensity is simply a constant...

$$d\lambda/d\mu = -\lambda_0/v_0 \text{ (note the linear relationship)}$$

- As previously implied, a value of  $\lambda_0$  and  $v_0$  must be determined prior to utilizing this model

Software Reliability  
© Jeff Knickerbocker, 2001

17

SW DER Conference 2001

### Musa Exponential Model

Failure Intensity vs. Total Number of Failures Example

It has been determined a software component is expected to experience 200 failures over an “infinite” period of time

Based on our experience, we know the initial failure intensity is 15 failures per *execution hour* (versus calendar time)

We have observed and resolved 75 faults

What the failure intensity after we have removed those 75 faults?

- $\lambda_0 = 15$  failures/execution hour
- $v_0 = 200$  failures
- $\mu = 75$  failures

$$\lambda(75) = 15(1 - 75/200) = 9.375 \sim 9 \text{ failures/execution hr}$$

with a **constant** decreasing failure slope of

$$d\lambda/d\mu = -\lambda_0/v_0 = -15/200 = -0.075 \text{ failures/execution hr}$$

(assuming faults are removed)

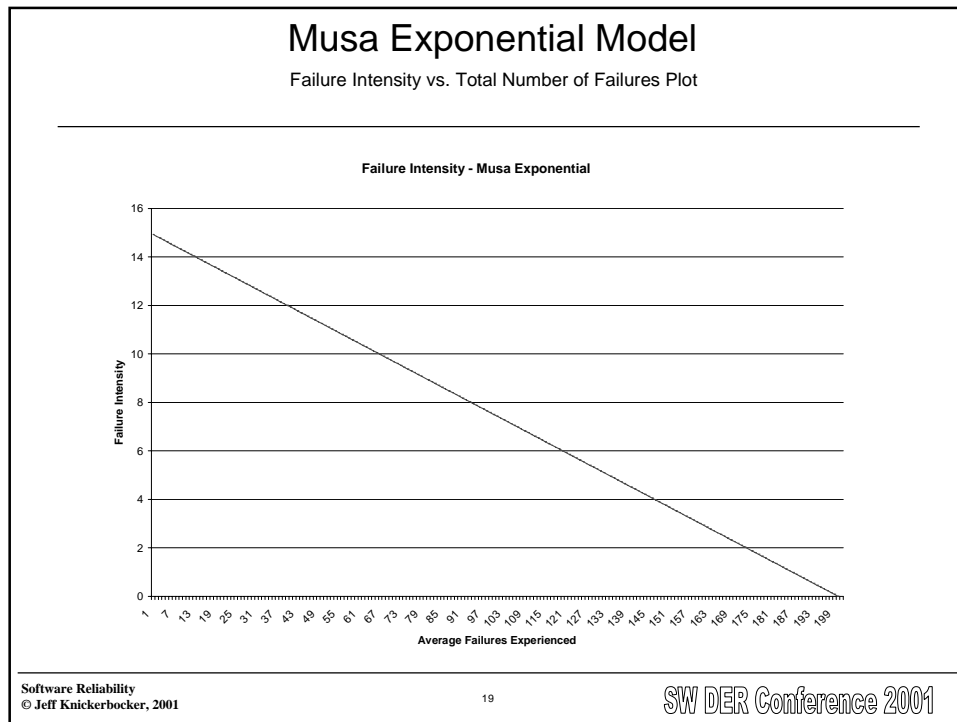
Software Reliability  
© Jeff Knickerbocker, 2001

18

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability



### Musa Exponential Model

Experienced Failures vs. Time Exponential Equation and Example

- This form of this equation is defined as follows:
 
$$\mu(T) = v_0(1 - e^{-(\lambda_0 T/v_0)})$$
- Using the same parameters as in the previous example,
 

Assume we are interested in how many failures we will experience after 10 hours of operation and again after 120 hours of operation assuming faults are removed as they are discovered  
Note this may be used as an indicator of remaining test time assuming continuing test effectiveness

T = 10 **execution** hours,  
 $\mu(10) = 200(1 - \exp(-(15 \cdot 10)/200)) = 200(1 - \exp(-3/4)) \sim 106$  failures

T = 120 **execution** hours,  
 $\mu(120) = 200(1 - \exp(-(15 \cdot 120)/200)) = 200(1 - \exp(-9)) \sim 200$  failures

This implies we will take care of more than half of the faults in the first 10 hours.  
Intuitively this is very interesting – we find the easy problems early and in great abundance but the later problems are harder and take longer!

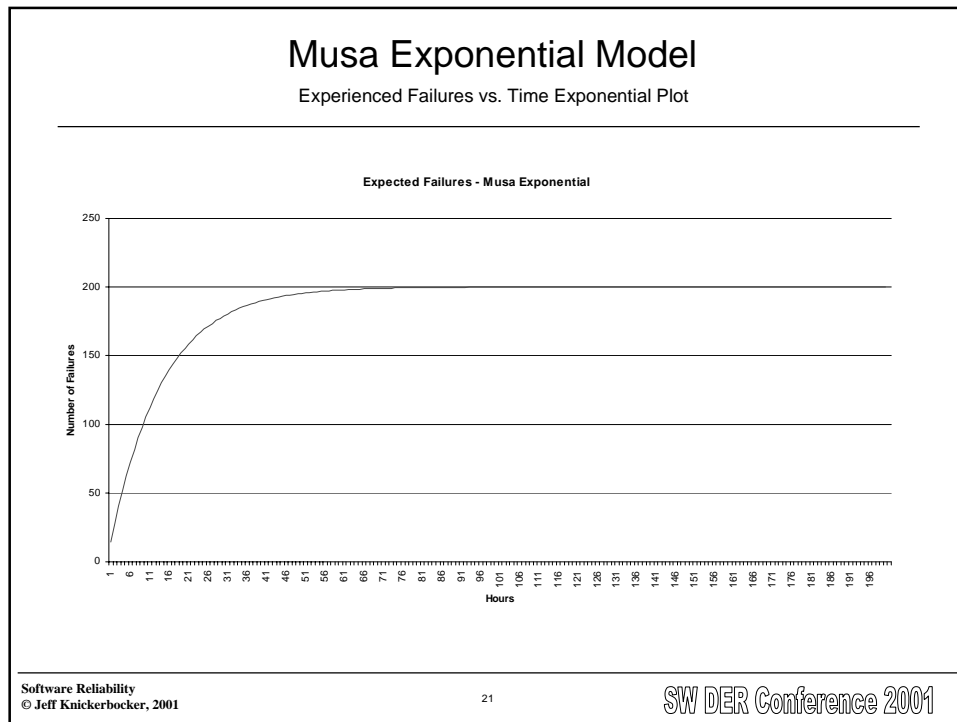
Software Reliability  
© Jeff Knickerbocker, 2001

20

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability



### Musa Exponential Model

Failure Intensity vs. Time Exponential Equation and Example

- This form of this equation is defined as follows:
 
$$\lambda(T) = \lambda_0 e^{(-\lambda_0 T / \nu_0)}$$
- Again, using the same parameters as in the previous examples,
 

Assume we are interested in what the failure intensity will be after 10 hours of operation and again after 120 hours of operation assuming faults are removed as they are discovered  
This may also be used as an indicator of remaining test time assuming continuing test effectiveness

T = 10 **execution** hours,  
 $\lambda(10) = 15(\exp(-15 \cdot 10 / 200)) = 15(\exp(-3/4)) \sim 7$  failures per execution hour

T = 120 **execution** hours,  
 $\lambda(120) = 15(\exp(-15 \cdot 120 / 200)) = 200(\exp(-9)) \sim 2$  failures per 1000 execution hours

This implies we will take care of many of the most prevalent faults in the first 10 hours of testing.  
 Again this is a nice intuitive piece of information. The easy, frequent faults are fixed early – the harder problems appear with much less regularity!

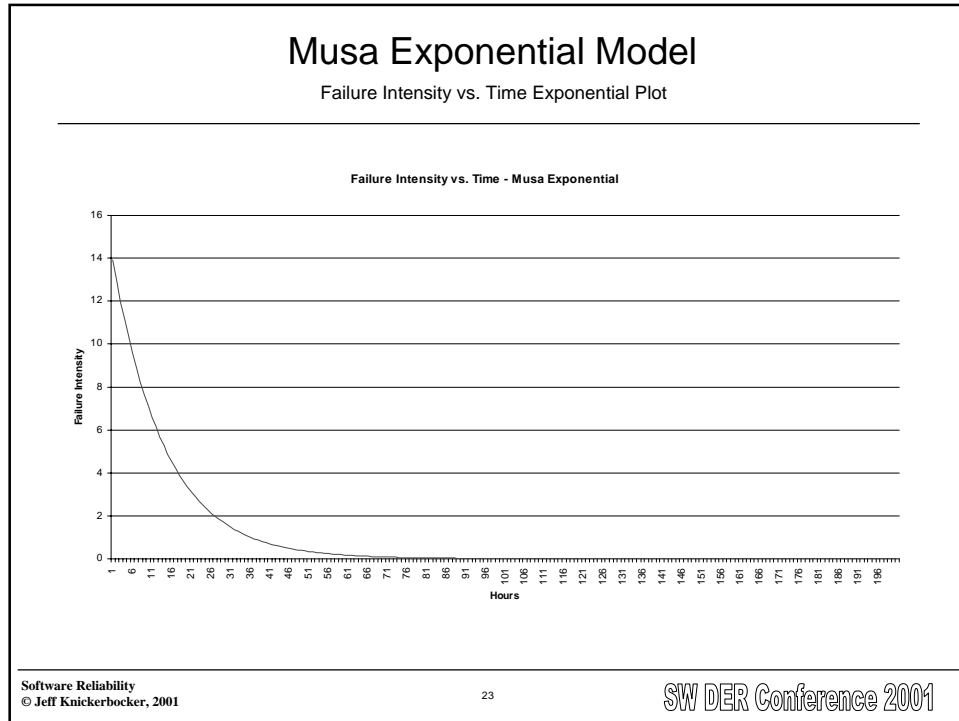
Software Reliability  
© Jeff Knickerbocker, 2001

22

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability



**Musa Exponential Model**  
Expected Failures to Reach a Specific Failure Intensity Equation and Example

---

- This form of this equation is defined as follows:
 
$$\Delta\mu = v_0/\lambda_0(\lambda_p - \lambda_F),$$
 where  
 $\lambda_p$  is the current failure intensity and  
 $\lambda_F$  is the future, desired failure intensity
- Again, using the same parameters as in the previous examples,  
 Assume we are interested in how many failures will occur before we reach a desired failure intensity of  $\lambda_F = 0.00345$  failures per execution hour assuming our present failure intensity  $\lambda_p = 0.01$  failures per execution hour  
 This may also be used as an indicator of remaining test time assuming continuing test effectiveness if we have an error removal rate estimated
 
$$\lambda_F = 0.00345,$$

$$\lambda_p = 0.01,$$

$$\Delta\mu = 200/15(0.01 - 0.00345) = 1.31 \text{ or at least two more failures need to be found}$$

Software Reliability  
© Jeff Knickerbocker, 2001

24

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Expected Time to Reach a Specific Failure Intensity Equation and Example

- This form of this equation is defined as follows:

$$\Delta T = v_0 / \lambda_0 \ln(\lambda_p / \lambda_f),$$

where

$\lambda_p$  is the current failure intensity and

$\lambda_f$  is the future, desired failure intensity

- Again, using the same parameters as in the previous examples,

Assume we are interested in how much time will pass before we reach a desired failure intensity of  $\lambda_f = 0.00345$  failures per execution hour assuming our present failure intensity  $\lambda_p = 0.01$  failures per execution hour

$$\lambda_f = 0.00345,$$

$$\lambda_p = 0.01,$$

$$\Delta T = 200/15(\ln(0.01 / 0.00345)) \sim 14.2 \text{ execution hours}$$

Software Reliability  
© Jeff Knickerbocker, 2001

25

SW DER Conference 2001

### Musa Exponential Model

Field Failure Intensity Equation

- This form of this equation is defined as follows:

$$R(T) = e^{-\lambda T}$$

where

$\lambda$  is the failure intensity at the time of release

- Again, using the same parameters as in the previous examples,

Assume we wish to know the reliability for a 10 hour mission with  $\lambda = 0.00345$  as noted in a previous example

$$R(10) = e^{-(0.00345 \cdot 10)} = 0.966 \text{ or } \sim 96.6\%$$

The form is identical to that of the simple ratio model presented earlier

Note the implications here - for 99.999% reliability on a 4 hour mission (typical commercial flight), the failure intensity needs to be less than 0.0000025 failures per hour as shown below:

$$\lambda = -\ln(0.99999)/4\text{hr} \sim 0.0000025 \text{ failures/hr}$$

Software Reliability  
© Jeff Knickerbocker, 2001

26

SW DER Conference 2001

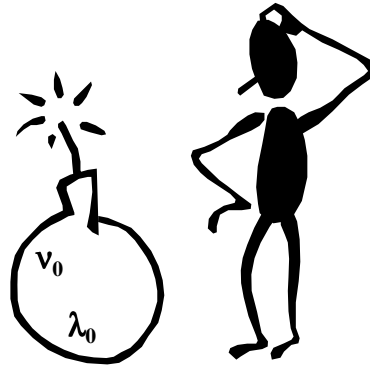
# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

About those empirical parameters...

- The model is very attractive but there are those empirical issues
- Parameter determination is most critical and also extremely difficult for this model
- The difficulty is not in the calculations!
- The parameters require combined **qualitative** and **quantitative** data that must be evaluated in order to determine the initial failure intensity rate ( $\lambda_0$ ) and the total number of failures ( $v_0$ )
- Several suggestions are outlined



Software Reliability  
© Jeff Knickerbocker, 2001

27

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Total Number of Failures ( $v_0$ )



- $v_0$  can be estimated by the following equation

$$v_0 = \omega_0 / \beta$$

where

$\beta$  is the fault reduction factor and  
 $\omega_0$  is the number of inherent faults

- Unfortunately rather than helping our current dilemma, we have complicated our lives by replacing one unknown variable with two!
- We **hope** that it will be easier to find the two component values rather than the total number of failures,  $v_0$

Software Reliability  
© Jeff Knickerbocker, 2001

28

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Parameter Estimation - Inherent Faults ( $\omega_0$ )

- Error seeding
  - ***Intentionally*** place faults in a software program
  - Track the total number of faults removed through testing
  - Using the ratio of seeded faults found to inherent faults found allows an estimate of the total inherent faults
  - The method works as follows:

Assume 20 faults are “seeded”, initial verification activities find 7 of the seeded faults, and 30 non-seeded or inherent faults are found

$$N_{\text{seeded}} = 20$$
$$N_{\text{seededFound}} = 7$$
$$N_{\text{inherent}} = 30$$
$$\omega_0 = (N_{\text{seeded}} / N_{\text{seededFound}}) * N_{\text{inherent}} = (20/7) * 30 = 600/7 \sim 86 \text{ inherent faults}$$
  - But the technique is not without problems...

Software Reliability  
© Jeff Knickerbocker, 2001

29

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Inherent Faults ( $\omega_0$ )

- Error seeding - the down side
  - Seeded faults may not be representative of the inherent faults - intentional faults are often much easier to find than unintentional faults
  - Knowledgeable staff is hard to find
    - » Staff most familiar with the program will be the best error-seeders
    - » The same staff may also be some of the most knowledgeable verification team members
    - » If a staff member is used for error seeding, that staff member will not be an alternative for working verification activities
  - Seeded errors must be tracked – software cannot be released that has intentional faults
  - Seeded faults may mask more serious inherent faults
  - Given these considerations, other approaches need to be investigated

Software Reliability  
© Jeff Knickerbocker, 2001

30

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Parameter Estimation - Inherent Faults ( $\omega_0$ )

- Development and maintenance of a fault database - preferred
  - Inherent errors are tracked over multiple programs within an organization
  - A ratio of faults to source lines of code (SLOCs or KSLOCs) may developed for each program (fault density,  $\rho$ )
  - Averaging fault density values will provide a scale factor that may be used to determine  $\omega_0$  as follows:

$$\rho_i = N_{\text{faults}_i} / \text{SLOC}_i$$
$$\rho_{\text{bar}} = (\rho_1 + \rho_2 + \dots + \rho_n) / n$$
$$\omega_0 = \text{SLOC}_{\text{current}} \rho_{\text{bar}}$$

- Issues?
  - » Data availability on a consistent basis (database maintenance)
  - » The approach will not be exact (neither is seeding)
  - » Current research indicates that for any given organization which develops a series of similar products, similar fault densities will exist across multiple programs unless some type of disruptive event is introduced (tools, training, massive staff changes, and so on)
  - » Tabular starting data may be available - Musa suggests a typical fault density will be 130 faults/KSLOC starting with the code phase immediately after successful compilation

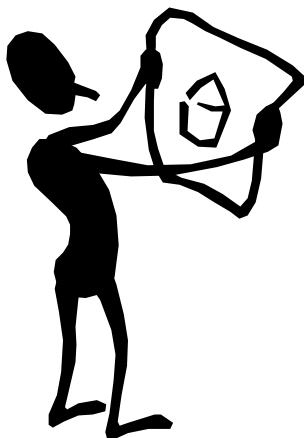
Software Reliability  
© Jeff Knickerbocker, 2001

31

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Fault Reduction Factor ( $\beta$ )



- $\beta$  can also be estimated...
  - $\beta$  is a ratio that compares the net fault reduction to the failures experienced
  - Initially would seem to be unity
  - Removing faults often causes faults
    - » Removing 100 faults may introduce 5 new faults
    - » This would lead to a net reduction of 95 faults or a fault reduction factor of 0.95 or 95%
- $\beta$  should be based on historical data...
  - Lacking that, current research would indicate that a fault reduction of 0.955 is average

Software Reliability  
© Jeff Knickerbocker, 2001

32

SW DER Conference 2001



# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Parameter Estimation - Closing on Total Number of Failures ( $v_0$ )

- $v_0$  can now be determined

$$v_0 = \omega_0 / \beta$$

where

$\beta$  is the fault reduction factor and

$\omega_0$  is the number of inherent faults

Using the numbers from the previous discussion,

$$\beta = 0.95$$

$$\omega_0 = 86 \text{ faults}$$

$$v_0 = 86 / 0.95 \sim 91 \text{ faults (rounding up)}$$



Software Reliability  
© Jeff Knickerbocker, 2001

33

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Initial Failure Intensity ( $\lambda_0$ )

- $\lambda_0$  can be estimated by the following equation

$$\lambda_0 = fK\omega_0$$

where

$f$  is the linear execution rate of the program (instructions per unit time)

$K$  is fault exposure ratio (accounts for iterative nature of programs and partially correct software)

$\omega_0$  inherent faults ( $v_0 = \omega_0 / \beta$ )

- Again we have complicated our lives by replacing one unknown variable by two!



Software Reliability  
© Jeff Knickerbocker, 2001

34

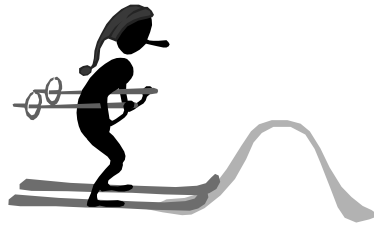
SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Parameter Estimation - Linear Execution Rate (f)

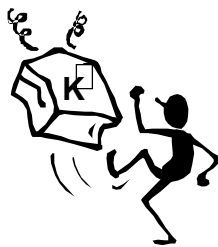


- f is an easy, quantitative ratio...

$$f = r/l$$

where

- r is the average object instruction rate
- l is the number of object instructions



- But then there is fault exposure ratio, K...
  - K addresses the issue of partially correct software
  - Sometimes it fails and sometimes it doesn't...

Software Reliability  
© Jeff Knickerbocker, 2001

35

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Fault Exposure Ratio (K)

- K should be determined from averaged historical data as follows...

$$K = \lambda_{0H} / \beta_H f v_0$$

where

- $\lambda_{0H}$  is the averaged initial failure intensity
- $\beta_H$  is the averaged fault reduction factor
- f is the linear execution rate
- $v_0$  is the total number of failures

- No historical data?
  - K is generally quit small - on the order of  $1.8 \times 10^{-7} - 11 \times 10^{-7}$
  - Efforts are underway to try and relate K to program structure and type (e.g., McCabe complexity combined with the application type)

Software Reliability  
© Jeff Knickerbocker, 2001

36

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

Parameter Estimation - Fault Exposure Ratio Example (K)

- Example solution for K

recall

$$K = \lambda_{0H} / \beta_H f v_0 \text{ and}$$
$$f = r/l$$

allow

$$f = (4 * 10^9 \text{ object instructions/s}) / (1 * 10^5 \text{ object instructions}) = 40,000 \text{ cycles/s}$$

$$\lambda_{0H} = 0.8 \text{ failures/s} \quad (\text{note change on time units from hours to seconds})$$

$$\beta_H = 0.95$$

$$v_0 = 200 \text{ failures}$$

then

$$K = (0.8) / (0.95 * (40,000) 200) = 1.05 * 10^{-7}$$

Software Reliability  
© Jeff Knickerbocker, 2001

37

SW DER Conference 2001

### Musa Exponential Model

Parameter Estimation - Initial Fault Fault Intensity Example ( $\lambda_0$ )

- Finally recall  $\lambda_0 = fK\omega_0$

where

f is the linear execution rate of the program  
(instructions per unit time)

K is fault exposure ratio (accounts for  
iterative nature of programs and partially  
correct software)

$\omega_0$  inherent faults ( $v_0 = \omega_0 / \beta$ )

Using our previous values,

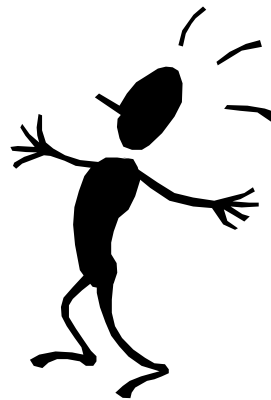
$$\lambda_0 = (40,000)(1.05 * 10^{-7})(86) = 0.362$$

failures/execution s

or

$$\lambda_0 \sim 1,300 \text{ failures/execution hr}$$

While this seems very large, we must remember we are starting the fault count immediately after successful compilation and we are really screaming through our code (assuming we can keep running). As we get better processes through our feedback mechanisms, our initial failure intensity may drop.



Software Reliability  
© Jeff Knickerbocker, 2001

38

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Musa Exponential Model

#### Strengths

- Simple to calculate (after the empirical parameters are found)
- Intuitively appealing
- Similar to the hardware exponential model
- Could be used to determine if the current project is fitting in organizational norms at various phases of development - e.g., fault intensity at unit test, integration test, system test, post-certification, ...
- Could be used for organizational “process improvement” measurements if adequate failure reporting systems are in place
- When combined with other safety/process oriented approaches, such as FHA/FMEACA, requirements coverage analysis and structural coverage analysis, the models could provide another perspective on the adequacy of the system for a specific mission role

Software Reliability  
© Jeff Knickerbocker, 2001

39

SW DER Conference 2001

### Musa Exponential Model

#### Concerns

- Not really quantitative - but sure looks like it initially
- Actually quantitative, qualitative, and iterative
- Development of values for initial failure intensity ( $\lambda_0$ ), failure intensity decay rate ( $\beta$ ), and the total number of faults expected ( $v_0$ ) is not trivial
- Naïve application of the models at arbitrary or different phases and possibly to different products will result in an apples to oranges comparisons
- Long-term commitment is required - Will the organization support or even tolerate such a change if it is not the current procedure?
- Other concerns are the same as those mentioned for the simple ratio model

Software Reliability  
© Jeff Knickerbocker, 2001

40

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability

### Observations/Recommendations

- Current application of software reliability models in the mission/safety critical arena could have the most direct impact on evaluating software components for re-use
  - » Mission needs and environments must be analyzed to determine if a particular component is a candidate for re-use, but if it is, software reliability predictions should be considered
  - » The initial operational profiles could provide insight about the suitability for re-use (did the operational profiles “cover the corners”?)
  - » Operational data needs to be used and is likely available in the air transport market - other sectors may not have such data readily available
  - » If possible, internal clocks should be used to track actual operational time, and possibly fault histories and categories
- Assuming problem tracking mechanisms have been in place over the life of the program, the simple ratio model could be very useful

Software Reliability  
© Jeff Knickerbocker, 2001

41

SW DER Conference 2001

### Conclusions

- Software reliability models are not trivial and are not a “silver bullet” for creating safe, reliable software
- Careful application of software reliability techniques, along with the requisite data collection and analysis required to support the models could assist the conscientious organization in developing a better product
- While the current models are not exact, they are providing the basis for future development - further research and refinement will be required
- Reliability models can provide another dimension to the evaluation of software products and software intensive systems
- Software reliability models do provide an opportunity to stand out among the various marketplace vendors

There is more to software reliability than models - check it out if you really have an interest in following this methodology

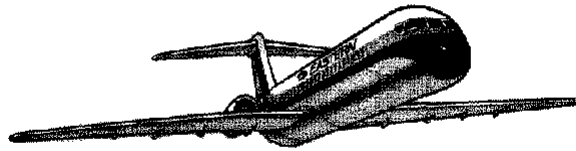
Software Reliability  
© Jeff Knickerbocker, 2001

42

SW DER Conference 2001

# FAA National Software Conference, June 2001

## Software Reliability



Software Reliability  
© Jeff Knickerbocker, 2001

43

SW DER Conference 2001